2, multitasking virtual machine 102 includes tasks 106 and 108 and shared runtime system 112. Shared runtime system 112 includes shared representation of class A 202, shared representation of class B 204, and task class mirror tables 206 and 208. In general, the portion of runtime system 112 that is shared among all tasks includes a shared representation of every loaded class, along with the associated task class mirror table.

[0035] Shared representation of class A 202, and shared representation of class B 204 include code (platform-independent code, and possibly, equivalent native code produced by the dynamic compiler of the virtual machine) and data (e.g., description of fields of instance and class variables, methods, interfaces, literal constants, symbolic links to other classes, etc.), which can be shared among any task requiring classes A and B. Task class mirror tables 206 and 208 are tables of pointers to task class mirrors for the respective classes. A task class mirror includes information for a class, which cannot be shared with other tasks as described below and in conjunction with FIG. 3. Task class mirror tables serve both as a multiplexing mechanism to retrieve the task class mirror of a class specific to a task, and a method for encoding the per-task initialization status of a class that enables fast implementation of class initialization barriers.

[0036] Data of a class that is specific to a task is stored in a task class mirror object, e.g., data of class A for task 106, respectively task 108, is stored in task class mirror 216, respectively, task class mirror 220. Example of class data that is specific to a class includes:

- the class own variables (e.g., the static variables of the class in the Java programming language),
- the class's initialization state,
- a reference to an object representing the class for programs (e.g., an instance of the class java.lang.Class in the case of the Java programming

language).

The references to the task class mirror objects of a class (e.g., task class mirror object 216 and 220) are stored in the task class mirror table associated with the shared representation of that class (e.g., task class mirror table 206 of shared representation of class A 202).

[0037] Each task is assigned a unique identifier by multitasking virtual machine 102. This unique identifier is used to compute an index to the task class mirror table of any shared representation of a class and obtain the corresponding task class mirror for that task. For example, the unique identifier of task 108 can be used to compute an index to task class mirror table 206 (respectively, task class mirror table 208) and obtain task class mirror 220 (respectively, task class mirror 224) that holds class A (respectively, class B) data that are specific to task 108.

Task Class Mirror Table

[0038] FIG. 3A illustrates task class mirror table 302 associated with the shared representation of a class C, before C's initialization but after its loading. Task class mirror table 302 is created when the shared representation of C is created, which happens typically upon the first request to load class C from any task. Task class mirror table 302 includes two entries per task, called respectively the resolved entry and the initialized entry. The present invention favors an arrangement where the entries assigned to a task are next to each other, so that the location of one entry can be simply found from the location of the other. Each entry holds the value of a pointer to a task class mirror object. All the entries of the task class mirror table are initialized to null upon its creation. Task class mirror table 302 includes sufficient entries to accommodate the expected number of tasks, however, the size of task class mirror table 302 can be dynamically adjusted during operation.

[0039] Upon loading of class C by a task, a task class mirror 304 is created and a pointer 306 to it is stored in the resolved entry of the task class mirror table 302. The initialized entry is left set to the null value. This configuration of the entries for a given task indicates that the task has loaded the class but has not initialized it yet.

[0040] FIG. 3B illustrates task class mirror table 302 after initialization in accordance with an embodiment of the present invention. A class initialization barrier determines if a class has been initialized for a specific task by examining the initialized entry of the class's task class mirror table for this task, as described above. If the pointer at that entry is null, multitasking virtual machine calls some runtime function that initializes the class on behalf of the task. Synchronizations against concurrent initializations of the same class by multiple threads of the same task are needed only in the runtime function that performs the actual initialization of the class, not in the barrier itself. The initialization completes by setting the initialized entry of task class mirror table 302 to a pointer 308 to the task class mirror 304 for this task. This prevents subsequent executions of the class initialization barrier by the task from calling multitasking virtual machine 102's runtime to initialize the class.

[0041] FIG. 3E shows the task class mirror table associated with the shared representation of a initialization-less class C when the multitasking virtual machine implementation treat them specially so as to minimize space consumption. In this case, the table has only one entry per task, and the entry is set upon loading of class C by the corresponding task. Classes that require initialization are dealt with as previously described, except a different arrangement of the entries of a task class mirror table must used, as shown on FIG. 3C and FIG. 3D. In other words, the only difference with the mechanisms